APPENDIX

```
#
#
# Value encodings:
#     ''    -  String (and asciiz is not appended)
#     ""    -  BMP string of ASCII characters
#     []    -  Hex octet string
#     <>    -  IP
#     {}    -  Object ID
#   Other -  Integer


1 system = 0
2  allocations = 0
3   vtPoolSize = 100000
3   vtNodeCount = 30000
3   channels = 800
3   chanDescs = 5
3   messages = 400
3   nameChans = 10
3   tpktChans = 25
3   udpChans = 5
3   protocols = 800
3   maxProcs = 800

1 RAS = 0
2  responseTimeOut = 20
2  allowCallsWhenNonReg = 0
2  manualRegistration = 0
2  manualDiscovery = 0
3   defaultGatekeeper = 0
4    ipAddress = 0
5     ip = [c8c8c8c8]
5     port = 1719

2  registrationInfo = 0
3   terminalType = 0
4    vendor = 0
5     vendor = 0
6      t35CountryCode = 11
6      t35Extension = 11
6      manufacturerCode = 11
5     productId = 'Test application'
5     versionId = 'RADVision'
4    terminal = 0
4    mc = 0
4    undefinedNode = 0
3   terminalAlias = 0
4    * = 0
5     e164 = '12345'

2  rasMulticastAddress = 0
3   ipAddress = 0
4    ip = <224.0.1.41>
4    port = 1718
2  rasPort = 0

1 Q931 = 0
2  manualAccept=0
2  responseTimeOut = 150
2  connectTimeOut = 500
2  callSignalingPort = 1720
2  maxCalls = 10

1 h245 = 0
2  masterSlave = 0
```

```
terminalType = 50
timeout = 100
capabilities = 0
 timeout = 100
 terminalCapabilitySet = 0
  sequenceNumber = 0
  protocolIdentifier = [00]
  multiplexCapability = 0
   h2250Capability = 0
    maximumAudioDelayJitter = 60
    receiveMultipointCapability = 0
     multicastCapability = 0
     multiUniCastConference = 0
     mediaDistributionCapability = 0
      * = 0
       centralizedControl = 0
       distributedControl = 0
       centralizedAudio = 0
       distributedAudio = 0
       centralizedVideo = 0
       distributedVideo = 0

    transmitMultipointCapability = 0
     multicastCapability = 0
     multiUniCastConference = 0
     mediaDistributionCapability = 0
      * = 0
       centralizedControl = 0
       distributedControl = 0
       centralizedAudio = 0
       distributedAudio = 0
       centralizedVideo = 0
       distributedVideo = 0

    receiveAndTransmitMultipointCapability = 0
     multicastCapability = 0
     multiUniCastConference = 0
     mediaDistributionCapability = 0
      * = 0
       centralizedControl = 0
       distributedControl = 0
       centralizedAudio = 0
       distributedAudio = 0
       centralizedVideo = 0
       distributedVideo = 0

    mcCapability = 0
     centralizedConferenceMC = 0
     decentralizedConferenceMC = 0
    rtcpVideoControlCapability = 0
    mediaPacketizationCapability = 0
     h261aVideoPacketization = 0

  capabilityTable = 0
   * = 0
    capabilityTableEntryNumber = 7111
    capability = 0
     receiveAudioCapability = 0
      g711Ulaw64k = 60

   * = 0
    capabilityTableEntryNumber = 7110
    capability = 0
     receiveAudioCapability = 0
      g711Alaw64k = 60

  * = 0
```

```
6       capabilityTableEntryNumber = 728
6       capability = 0
7        receiveAudioCapability = 0
8         g728 = 60

5       * = 0
6       capabilityTableEntryNumber = 261
6       capability = 0
7        receiveVideoCapability = 0
8         h261VideoCapability = 0
9          qcifMPI = 1
9          cifMPI = 1
9          temporalSpatialTradeOffCapability = 0
9          maxBitRate = 600
9          stillImageTransmission = 0

5       * = 0
6       capabilityTableEntryNumber = 263
6       capability = 0
7        receiveVideoCapability = 0
8         h263VideoCapability = 0
9          sqcifMPI = 1
9          qcifMPI = 1
9          cifMPI = 1
9          maxBitRate = 1000
9          unrestrictedVector = 0
9          arithmeticCoding = 0
9          advancedPrediction = 0
9          pbFrames = 0
9          temporalSpatialTradeOffCapability = 0
9          errorCompensation = 0

5       * = 0
6       capabilityTableEntryNumber = 7231
6       capability = 0
7        receiveAudioCapability = 0
8         g7231 = 0
9          maxAl-sduAudioFrames = 8
9          silenceSuppression = 0


5       * = 0 # Sequence
6       capabilityTableEntryNumber = 120 # INTEGER [1..65535]
6       capability = 0
7        receiveAndTransmitDataApplicationCapability = 0 # Sequence
8         application = 0
9          t120 = 0
10          separateLANStack = 0 # NULL [0..0]
8         maxBitRate = 1000 # INTEGER [0..4294967295]


4     capabilityDescriptors = 0
5      * = 0
6       capabilityDescriptorNumber = 0
6       simultaneousCapabilities = 0
7        * = 0
8         * = 7111
8         * = 7110
8         * = 7231
8         * = 728
7        * = 0
8         * = 261
8         * = 263
7        * = 0
8         * = 120
```

```
channels = 0
 * = 0
  name = 'g711Alaw64k'
  dataType = 0
   audioData = 0
    g711Alaw64k = 60

 * = 0
  name = 'g711Ulaw64k'
  dataType = 0
   audioData = 0
    g711Ulaw64k = 60

 * = 0
  name = 'g728'
  dataType = 0
   audioData = 0
    g728 = 60

 * = 0
  name = 'g7231'
  dataType = 0
   audioData = 0
    g7231 = 0
     maxAl-sduAudioFrames = 8
     silenceSuppression = 0

 * = 0
  name = 'h261VideoCapability'
  dataType = 0
   videoData = 0
    h261VideoCapability = 0
     qcifMPI = 1
     cifMPI = 1
     temporalSpatialTradeOffCapability = 0
     maxBitRate = 600
     stillImageTransmission = 0

 * = 0
  name = 'h263VideoCapability'
  dataType = 0
   videoData = 0
    h263VideoCapability = 0
     sqcifMPI = 1
     qcifMPI = 1
     cifMPI = 1
     maxBitRate = 1000
     unrestrictedVector = 0
     arithmeticCoding = 0
     advancedPrediction = 0
     pbFrames = 0
     temporalSpatialTradeOffCapability = 0
     errorCompensation = 0

 * = 0
  name = 't120'
  dataType = 0
   data = 0
    application = 0
     t120 = 0
      separateLANStack = 0
    maxBitRate = 1000

[MWIACTIVATE_SR]
```

```
servedUserNr=serveduser@example.com
basicService=0815
msgCentreId=4711
nbOfMessages=1
originatingNr=originating@example.com
timestamp=19970621194530
priority=7

[MWIDEACTIVATE_SR]
servedUserNr=serveduser@example.com
basicService=0815
msgCentreId=4711
callbackReq=1

[MWIINTERROGATE_SR]
servedUserNr=serveduser@example.com
basicService=0815
msgCentreId=4711
callbackReq=1

[MWIACK_SR]
basicService=0815
msgCentreId=4711
nbOfMessages=1
originatingNr=originating@example.com
timestamp=19970621194530
priority=7

[MWIREJ_SR]
Error=8
```

```
#include <rvcommon.h>
#include <msg.h>
#include <ms.h>
#include <pdlproc.h>
#include <stkutils.h>
#include <rtree.h>
#include <pdlraw.h>
#include <cmintr.h>
#include "h450api.h"
#include "h450struct.h"


static LOCAL_STRUCT_PTR control;
static cmElem* app;
static char string[2048];     //string, which carries the H.450
message
static unsigned int n;                    // counter to step through
the string


RVAPI void CALLCONV initAPI(MYHANDLE *ahandle)
{
    control = (LOCAL_STRUCT_PTR)calloc(1, sizeof(LOCAL_STRUCT));
    *ahandle = (MYHANDLE *)control;
}


RVAPI void CALLCONV endAPI()
{
      if (control)
      {
            free(control);
      }
}
```

```
RVAPI void CALLCONV setCallback(MYHANDLE ahandle, CALLBACK_T
callback)
{
LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)ahandle;

    local->callback = callback;
}


RVAPI void CALLCONV initH450Root(char * configfile, MYHANDLE
*rootshandle)
{
      char pdlName[280] = "*RADVH450";
      pdlInitNumbers numbers =
    {
       /*vtNodeCount    */2700,
       /*channels       */100,
       /*chanDescs  5    */5,
       /*messages       */50,
       /*tpktChans      */5,
       /*udpChans       */3,
       /*protocols      */40,
       /*maxProcs       */40,
      /*maxBuffSize    */2048,
    };
      app=(cmElem*)calloc(1,sizeof(cmElem));


      msOpen();

      app->hCfg=ciConstruct(configfile);
      ciGetString(app-
>hCfg,"system.pdlName",pdlName,sizeof(pdlName));


      pdlInit((HPdlAProtocol)app,&(app-
>hsProtocol),(newProtocolEH)newH450RootChild,( char*)pdlName,app-
>hCfg,&numbers);

      *rootshandle=(MYHANDLE*)(app->hsProtocol);


      pdlStart(app->hsProtocol);
}


RVAPI void CALLCONV endH450Root()
{
      if (app)
    {
            pdlEnd(app->hsProtocol);
            ciDestruct(app->hCfg);
            msClose();
            free(app);
    }
}


RVAPI void CALLCONV createH450protocol(char * machinename, CALLBACK_T
rootshandle, CALLBACK_T ahandle, MYHANDLE *h450shandle)
// create a new protocol under the root
{
      cmElem* app=(cmElem*)rootshandle;
```

```
        // create new protocol: runs the create block of the specified
machine
        pdlCreateProtocol((HPdlSProtocol)app,(HPdlAProtocol)ahandle,
        (HPdlSProtocol*)h450shandle, machinename,
(newProtocolEH)newH450Child,(newMessageEH)newH450Message,FALSE);
}


RVAPI void CALLCONV closeH450protocol(CALLBACK_T mglinkershandle)
{
        // close the protocol for the messagelinker
        pdlCloseProtocol((HPdlSProtocol)mglinkershandle);
}

int newH450Message(HPdlAProtocol haProtocol, HPdlSProtocol
hsProtocol, int   message)
{
        HPST synH;
        HPVT valH;
        int stNodeId;           // the syntaxtree nodeId
        int fieldId;            // the fieldId of the node
        int vtNodeId;           // the valuetree nodeId
        char rootname[256]; // the rootname of the syntaxtree root node

        LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)haProtocol;
        memset(string,0,sizeof(string));


        valH = pdlGetValTree((HPdlSProtocol)hsProtocol);


        vtNodeId=pvtGetNodeIdByPath(valH, message,"");

              synH=pvtGetSynTree(valH,vtNodeId);


        pvtGet(valH,vtNodeId,NULL,&stNodeId,NULL,NULL);


        pstGetRootName(synH, sizeof(rootname), rootname);


        fieldId=pstGetFieldId(synH, rootname);

        strcpy(string, "############################## H450-Message
##############################");
        local->callback(string);
        strcpy(string, "");


        recursion(haProtocol, valH, synH, stNodeId, vtNodeId, fieldId,
0);

        // enable to send message in on esting: local-
>callback(string);

        return 0;

}


char recursion(HPdlAProtocol haProtocol, HPVT valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab)

{
        LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)haProtocol;
```

```
        pstChild child;
        char fieldName[256];
        char fieldString[256];
        int stNodeIdChild;      // the syntaxtree nodeId of the Child
        int vtNodeIdChild;      // the valuetree nodeId of the Child
        int value;              // the value of the number stored in
the node or the leghth of the string stored in the node
        BOOL isString;          // TRUE if a string is stored in the
node
        int i;                          // counter

        vtNodeIdChild=vtNode;
        stNodeIdChild=stNode;

        // get the value stored in the node
        pvtGet(valH,vtNodeIdChild,NULL,NULL,&value,&isString);


        // insert the tabs
        for (i=0 ; i<tab; i++)
        {
              strcat(string, "    ");
        }

        // get the fieldname of the syntaxtree node
        pstGetFieldName(synH, fieldId, sizeof(fieldName), fieldName);

        strcat(string, "<");
        strcat(string, fieldName);
        strcat(string, ">");

        if (value<=0)
        {
              // Begin: remove to send message in one string
              local->callback(string);
              strcpy(string, "");
              // End: remove to send message in one string

        }

        // reads out the value, if a value is stored in the valuetree
node
        if (value>0)
                    {
                          // read out the string, if a string is stored
in the node
                          if (isString == TRUE)
                          {
                                pvtGetString(valH, vtNodeIdChild,
value, fieldString);
                                strcat(string, fieldString);
                          }

                          // read out the number, if a number is stored
in the node
                          else
                          {
                                // convert number to string
                                _itoa(value, fieldString, 10);
                                strcat(string, fieldString);
                          }
                          local->callback(string);
                          strcpy(string, "");
                    }


        // if there are any children of the current node , the function
```

```
        // calls itself with the stNode, vtNode and the fieldId of each
child, and with
        // tab increased by one
        if (pstGetNumberOfChildren(synH, stNodeIdChild)>0)
            {

                for (i=1; i<pstGetNumberOfChildren(synH,
stNodeIdChild)+1; i++)
                    {
                        pstGetChild(synH, stNodeIdChild, i, &child );

                        // function only calls itself, when the child
exists in the valuetree
                        if (pvtGetChild(valH, vtNode, child.fieldId,
&vtNodeIdChild)>0)
                        {
                            recursion(haProtocol, valH, synH,
child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
                        }
                    }

            }

        // if NodeType is Sequence of or Set of, the function also
searches for
        // children of the Sequence of XXX / Set of XXX node.
        if (pstGetNodeType(synH, stNode)==19 || pstGetNodeType(synH,
stNode)==20)
            {
            stNodeIdChild=pstGetNodeOfId(synH, stNode);
            vtNode=pvtChild(valH, vtNode);

            for (i=1; i<pstGetNumberOfChildren(synH,
stNodeIdChild)+1; i++)
                {
                    pstGetChild(synH, stNodeIdChild, i, &child );

                    // function only calls itself, when the child
exists in the valuetree
                    if (pvtGetChild(valH, vtNode, child.fieldId,
&vtNodeIdChild)>0)
                    {
                        recursion(haProtocol, valH, synH,
child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
                    }
                }
            }

        // insert the tabs
        for (i=0 ; i<tab; i++)
        {
            strcat(string, "    ");
        }

        strcat(string, "</");
        strcat(string, fieldName);
        strcat(string, ">");

        // Begin: remove to send message in one string
        local->callback(string);
        strcpy(string, "");
        // End: remove to send message in one string

        return 0;
}
```

```
int newH450RootChild(
          IN      HPdlAProtocol           haProtocolParent,
          IN      HPdlSProtocol           hsProtocol,
          OUT     HPdlAProtocol*          lphaProtocol,
          IN      char*                protocolName,
          OUT     newProtocolEH*          newSessionP,
          OUT     newMessageEH*           newCallMsgP,
          OUT     BOOL*                selfDestructing)
{
    *newSessionP=(newProtocolEH)newH450RootChild;
    *newCallMsgP=newH450Message;
    *lphaProtocol=haProtocolParent;
    *selfDestructing=TRUE;
    return 0;
}


int newH450Child(
          IN      HPdlAProtocol           haProtocolParent,
          IN      HPdlSProtocol           hsProtocol,
          OUT     HPdlAProtocol*          lphaProtocol,
          IN      char*                protocolName,
          OUT     newProtocolEH*          newSessionP,
          OUT     newMessageEH*           newCallMsgP,
          OUT     BOOL*                selfDestructing)
{
    *newSessionP=(newProtocolEH)newH450Child;
    *newCallMsgP=newH450Message;
    *lphaProtocol=haProtocolParent;
    *selfDestructing=TRUE;
    return 0;
}


RVAPI void CALLCONV sendmessageH450(char * message, CALLBACK_T
h450shandle)

{
      int nodeId;
      cmElem*
app=(cmElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

      // get the handle of the valuetree
      app->hVal = pdlGetValTree(app->hsProtocol);

      // create a new value tree based on "h4507app2pdl"
      nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
>hsProtocol,(char*)"h4507app2pdl"),0,NULL);
      n=0; /* set counter to 0*/

      // call function to fill in the fields of the valuetree sent in
message
      fillin(message, "", app->hVal, nodeId);
      pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}


char fillin (char * message, char * rootpath, HPVT hVal, int nodeId)
{
int length;
int nodeIdTemp;
char path[2048];
char fieldvalue[256];
memset(path,0,sizeof(path));
memset(fieldvalue,0,sizeof(fieldvalue));
```

```
strcpy(path, rootpath);
length = strlen(message);

while (message[n] == '<' && message[n+1] != '/')
{
      n++;
      fillin(message, path, hVal, nodeId);
      if (message[n] == 0 || (message[n] == '<' && message[n+1] ==
'/')) return 0;
}


      if (message[n] != '/')
      {
            n--;
            if (n!=0) strcat(path, ".");
            n++;
            while (message[n] != '>')
            {
                  strncat(path, message+n, 1);
                  n++;
            }
            n++;

            if (message[n] != '<')
            {
                  strcpy(fieldvalue, "");
                  while (message[n] != '<')
                  {
                        strncat(fieldvalue, message+n, 1);
                        n++;
                  }
                  nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,sizeof(fieldvalue),fieldvalue);
                  n++;
            }

            else
            {
                  nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,0,NULL);
                  if (message[n] == '<' && message[n+1] != '/')
                  {
                        fillin(message, path, hVal, nodeId);
                        n++;
                  }
                  else n++;
            }
      }

      if (message[n] == '/')
      {
            n--;
            while (message[n] != '>')
            {
                  n++;
            }
            n++;
            return 0;
      }
}
#ifndef H450API_H
#define H450API_H

#include <cm.h>
```

```
/* Typedefs */

typedef void *MYHANDLE;

typedef void (*CALLBACK_T)(char*);

typedef struct
{
    CALLBACK_T callback;
} LOCAL_STRUCT, *LOCAL_STRUCT_PTR;

/* Function prototypes */

RVAPI void CALLCONV initAPI(MYHANDLE *ahandle);
RVAPI void CALLCONV endAPI();
RVAPI void CALLCONV setCallback(MYHANDLE ahandle, CALLBACK_T
callback);
RVAPI void CALLCONV initH450Root(char * configfile, MYHANDLE
*rootshandle);
RVAPI void CALLCONV endH450Root();
RVAPI void CALLCONV sendmessageH450(char * string, CALLBACK_T
h450shandle);
RVAPI void CALLCONV initializeH(void);
RVAPI void CALLCONV createH450protocol(char * machinename, CALLBACK_T
rootshandle, CALLBACK_T ahandle, MYHANDLE *h450shandle);
RVAPI void CALLCONV closeH450protocol(CALLBACK_T mglinkershandle);

#endif /* H450API_H */
#ifndef H450_H
#define H450_H

#include <cm.h>


/* Typedefs */

typedef struct
{
    HRTREE      hrTree;  /* the tree of protocols*/
    HDRV            hTpktChan;/* ther are 2 types of external
channels */
    HDRV            hUdpChan;
    HPDLCHANDESC    hApplChan;/* application channel internal
channel,implemented by pdlapi*/
    HPDL            hPdl;     /* pointer to global structure in
pdlproc layer*/
    HCHN            hChn;    /* handle passed to all channels
functions*/
    HVALT           hVal;
    newCallbackEH   newCallback; /* global callback,that calls before
all callbacks
        created for X */
    setAppBufferEH  setAppBuffer;/* notify the appl about new message
on the one of the
        application channels*/
    HCFG            hCfg; /* configuration handle */
    HPDLRAW         hRaw; /* handle to pdl file */
    int               msa; /* pdlapi printing */
} pdlApp; /* one for all protocols contains all global values*/

typedef struct
{
    pdlApp*       app;
    int               id; /*id in the protocol tree  */
    HPdlAProtocol haProtocol; /* application handle*/
    HPDLP         hProc; /* pointer to machine */
```

```
    UINT32              key;  /*protocol parameter is stored as tree
*/
    newProtocolEH       newProtocol;/* called when new child protocol
is created*/
    newMessageEH  newMessage; /* protocol receives messages from
machine via this cllback */
    BOOL                passive; /* set if tried to close protocol
and not succeded because of childs*/
    BOOL                selfDestructing;/* can parent close child
protocol without closing this child from
                                application*/
} pdlElem; /*protocol structure */

/* Function prototypes */

int newH450RootChild(
          IN      HPdlAProtocol           haProtocolParent,
          IN      HPdlSProtocol           hsProtocol,
          OUT     HPdlAProtocol*          lphaProtocol,
          IN      char*             protocolName,
          OUT     newProtocolEH*          newSessionP,
          OUT     newMessageEH*           newCallMsgP,
          OUT     BOOL*             selfDestructing


          );

int newH450Child(
          IN      HPdlAProtocol           haProtocolParent,
          IN      HPdlSProtocol           hsProtocol,
          OUT     HPdlAProtocol*          lphaProtocol,
          IN      char*             protocolName,
          OUT     newProtocolEH*          newSessionP,
          OUT     newMessageEH*           newCallMsgP,
          OUT     BOOL*             selfDestructing


          );

int newH450Message(
             IN   HPdlAProtocol           haProtocol,
             IN   HPdlSProtocol           hsProtocol,
             IN   int                  message
             );
char recursion(HPdlAProtocol haProtocol, HPVT valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab);
char fillin (char * dummy, char * dummypath, HPVT hVal, int nodeId);

#endif /* H450_H *//

#include <rvcommon.h>
#include <msg.h>
#include <ms.h>
#include <pdlproc.h>
#include <stkutils.h>
#include <rtree.h>
#include <pdlraw.h>
#include <cmintr.h>
#include <emanag.h>
#include "h450api.h"
#include "h450struct.h"

#include <ssintr.h>


static LOCAL_STRUCT_PTR control;
```

```
static ssElem* app;
static char string[2048];
static char buffer[2048];
static unsigned int n;


RVAPI void CALLCONV tpInitAPI(MYHANDLE *ahandle)
{
    control = (LOCAL_STRUCT_PTR)calloc(1, sizeof(LOCAL_STRUCT));
    *ahandle = (MYHANDLE *)control;
}


RVAPI void CALLCONV tpEndAPI()
{
      if (control)
      {
            free(control);
      }
}


RVAPI void CALLCONV tpSetCallback(MYHANDLE ahandle, CALLBACK_T
callback)
{
LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)ahandle;

    local->callback = callback;
}
RVAPI int tpNewH450Message(HPdlAProtocol haProtocol, HPdlSProtocol
hsProtocol, int   message)
{
      HPST synH;
      HPVT valH;
            int stNodeId;
      int fieldId;
      int vtNodeId;                    char rootname[256];
      LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)haProtocol;
      memset(string,0,sizeof(string));
      memset(buffer,0,sizeof(buffer));

            valH = pdlGetValTree((HPdlSProtocol)hsProtocol);


      //vtNodeId=pvtGetNodeIdByPath(valH,
message,"h4501SupplemetaryService"); /* insert for h.450.1 */
      vtNodeId=pvtGetNodeIdByPath(valH, message,"");/* remove for
h.450.1 */

            synH=pvtGetSynTree(valH,vtNodeId);

            pvtGet(valH,vtNodeId,NULL,&stNodeId,NULL,NULL);

            pstGetRootName(synH, sizeof(rootname), rootname);


      fieldId=pstGetFieldId(synH, rootname);

      strcpy(string, "############  received H450-Message
############");/* remove for h.450.1 */
      local->callback(string);/* remove for h.450.1 */
      strcpy(string, "");

      //emEncode((HVALT)valH,vtNodeId,(BYTE*)buffer,sizeof(buffer),&e
ncoded); /* insert for h.450.1 */
      //local->callback(buffer); /* insert for h.450.1 */
```

```
        tpReadOutVT(haProtocol, valH, synH, stNodeId, vtNodeId,
fieldId, 0); /* remove for h.450.1 */

        // enable to send message in one string: local-
>callback(string);

        return 0;

}

RVAPI int CALLCONV tpCreateService(
                                IN      HSSAPP      hSSApp,
                                OUT     HSSSERVICE* hSSServ,
                                IN      HSSAPPSERVICE hSSaServ,
                        IN              char*   serviceName)
{
    int ret;
    ssElem* app=(ssElem*)hSSApp;

    ret = pdlCreateProtocol(app->hsProtocol,(HPdlAProtocol)hSSaServ,
      (HPdlSProtocol*)hSSServ,serviceName,
NULL,tpNewH450Message,FALSE);
    return ret;
}


char tpReadOutVT(HPdlAProtocol haProtocol, HPVT valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab)

{
      LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)haProtocol;
      pstChild child;
      char fieldName[256];
      char fieldString[256];
      int stNodeIdChild;
      int vtNodeIdChild;
      int value;
      BOOL isString;
      int i;

      vtNodeIdChild=vtNode;
      stNodeIdChild=stNode;


      pvtGet(valH,vtNodeIdChild,NULL,NULL,&value,&isString);


            for (i=0 ; i<tab; i++)
      {
            strcat(string, "    ");
      }

            pstGetFieldName(synH, fieldId, sizeof(fieldName),
fieldName);

      strcat(string, "<");
      strcat(string, fieldName);
      strcat(string, ">");



      if (value>0 && pstGetNodeType(synH, stNode)<15) //value>=0 ???
      {

            if (isString == TRUE)
```

```
                {
                        pvtGetString(valH, vtNodeIdChild, value,
fieldString);
                        strncat(string, fieldString, value);
            strcpy(fieldString, "");
                }


                else
                {

                        _itoa(value, fieldString, 10);
                        strcat(string, fieldString);
            strcpy(fieldString, "");
                }
                local->callback(string);
                strcpy(string, "");
        }
    else
        {

                local->callback(string);
                strcpy(string, "");

        }



        if (pstGetNumberOfChildren(synH, stNodeIdChild)>0)
                {

                for (i=1; i<pstGetNumberOfChildren(synH,
stNodeIdChild)+1; i++)
                        {
                                pstGetChild(synH, stNodeIdChild, i, &child );

                                                        if (pvtGetChild(valH,
vtNode, child.fieldId, &vtNodeIdChild)>0)
                                {
                                        tpReadOutVT(haProtocol, valH, synH,
child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
                                }
                        }

                }


        if (pstGetNodeType(synH, stNode)==19 || pstGetNodeType(synH,
stNode)==20)
                {
                stNodeIdChild=pstGetNodeOfId(synH, stNode);
                vtNode=pvtChild(valH, vtNode);

                for (i=1; i<pstGetNumberOfChildren(synH,
stNodeIdChild)+1; i++)
                        {
                                pstGetChild(synH, stNodeIdChild, i, &child );

                                                        if (pvtGetChild(valH,
vtNode, child.fieldId, &vtNodeIdChild)>0)
                                {
                                        tpReadOutVT(haProtocol, valH, synH,
child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
                                }
                        }
                }
```

```
            for (i=0 ; i<tab; i++)
        {
            strcat(string, "    ");
        }

        strcat(string, "</");
        strcat(string, fieldName);
        strcat(string, ">");

            local->callback(string);
        strcpy(string, "");

        return 0;
}

RVAPI void CALLCONV tpSendMessageH4507(char * message, HSSSERVICE
h450shandle)

{
        int nodeId;
        ssElem*
app=(ssElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

            app->hVal = pdlGetValTree(app->hsProtocol);

            nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
>hsProtocol,(char*)"h4507app2pdl"),0,NULL); /* remove for h.450.1 */

        n=0; /* set counter to 0*/


        tpPopulateVT(message, "", app->hVal, nodeId);
        pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}

RVAPI void CALLCONV tpSendMessageH4501(char * message, HSSSERVICE
h450shandle)

{
        int nodeId;
        ssElem*
app=(ssElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

            app->hVal = pdlGetValTree(app->hsProtocol);


        nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
>hsProtocol,(char*)"h4501AppPrimitive"),0,NULL); /* insert for
h.450.1 */

        n=0; /* set counter to 0*/


        tpPopulateVT(message, "", app->hVal, nodeId);
        pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}

RVAPI void CALLCONV tpSendMessageH450x(char * message, HSSSERVICE
h450shandle)

{
        int nodeId;
        ssElem*
app=(ssElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

            app->hVal = pdlGetValTree(app->hsProtocol);
```

```
            nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
>hsProtocol,(char*)"h450xAppPrimitive"),0,NULL); /* insert for
h.450.1 */

      n=0; /* set counter to 0*/


      tpPopulateVT(message, "", app->hVal, nodeId);
      pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}

int chrn2bmp1(char *str, int maxStrLen, BYTE* bmpStr)
{
    int i, i2;

    for (i = 0, i2 = 0; i < maxStrLen; i++, i2 += 2)
    {
      bmpStr[i2 + 0] = 0;
      bmpStr[i2 + 1] = str[i];
    }

    return i2;
}

int chr2bmp1(char *str, BYTE* bmpStr)
{
    return chrn2bmp1(str, strlen(str), bmpStr);
}


char tpPopulateVT (char * message, char * rootpath, HPVT hVal, int
nodeId)
{
UINT32 fieldvalueOctet;
int length;
int fieldvaluesize;
int fieldvalueInt;
int fieldvaluelength;
int bmpLength;
int nodeIdTemp;
char bmpStr[512];
char path[2048];
char fieldvalue[256];
char stringOrInt;
memset(path,0,sizeof(path));
memset(fieldvalue,0,sizeof(fieldvalue));
strcpy(path, rootpath);
length = strlen(message);

while (message[n] == '<' && message[n+1] != '/')
{
      n++;
      tpPopulateVT(message, path, hVal, nodeId);
      if (message[n] == 0 || (message[n] == '<' && message[n+1] ==
'/')) return 0;
}


      if (message[n] != '/')
      {
            n--;
            if (n!=0) strcat(path, ".");
            n++;
            while (message[n] != '>')
            {
                  strncat(path, message+n, 1);
                  n++;
```

```
                }
                n++;

                if (message[n] != '<')
                {
                        stringOrInt=message[n];
                        n++;
                        fieldvaluesize=0;
                        strcpy(fieldvalue, "");
                        while (message[n] != '<')
                        {
                                strncat(fieldvalue, message+n, 1);
                                n++;
                                fieldvaluesize++;
                        }
                        fieldvalue[fieldvaluesize]='\0';
                        fieldvaluesize++;
                        fieldvaluelength=strlen(fieldvalue);
                        if (stringOrInt=='s')
                        {
                                nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,fieldvaluelength,fieldvalue);
                        }
                        if (stringOrInt=='b')
                        {
                                bmpLength=chr2bmpl(fieldvalue,
(BYTE*)bmpStr);
                                nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,bmpLength,bmpStr);
                        }
                        if (stringOrInt=='i')
                        {
                                fieldvalueInt = atoi(fieldvalue);
                                nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,fieldvalueInt,NULL);
                        }
                        if (stringOrInt=='o')
                        {
                                fieldvalueOctet = atoi(fieldvalue);
                                nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,sizeof(UINT32),(fieldvalueOctet)?((ch
ar*)&fieldvalueOctet):(char*)"\0\0\0");
                        }
                        n++;
                }

                else
                {
                        nodeIdTemp =
pvtBuildByPath(hVal,nodeId,path,0,NULL);
                        if (message[n] == '<' && message[n+1] != '/')
                        {
                                tpPopulateVT(message, path, hVal, nodeId);
                                n++;
                        }
                        else n++;
                }
        }

        if (message[n] == '/')
        {
                n--;
                while (message[n] != '>')
                {
                        n++;
                }
                n++;
```

```
            return 0;
        }
}

#ifndef H450API_H
#define H450API_H

#include <cm.h>
#include <pdlapi.h>
#include <h450.h>


/* Typedefs */

typedef void *MYHANDLE;

typedef void (*CALLBACK_T)(char*);

typedef struct
{
    CALLBACK_T callback;
} LOCAL_STRUCT, *LOCAL_STRUCT_PTR;


RVAPI void CALLCONV tpInitAPI(MYHANDLE *ahandle);
RVAPI void CALLCONV tpEndAPI();
RVAPI void CALLCONV tpSetCallback(MYHANDLE ahandle, CALLBACK_T
callback);
RVAPI void CALLCONV tpSendMessageH4501(char * string, HSSSERVICE
h450shandle);
RVAPI void CALLCONV tpSendMessageH450x(char * string, HSSSERVICE
h450shandle);
RVAPI void CALLCONV tpSendMessageH4507(char * string, HSSSERVICE
h450shandle);
RVAPI int  tpNewH450Message(HPdlAProtocol haProtocol, HPdlSProtocol
hsProtocol, int   message);
RVAPI int  CALLCONV tpCreateService(HSSAPP hSSApp, HSSSERVICE*
hSSServ, HSSAPPSERVICE hSSaServ, char*    serviceName);

#endif /* H450API_H */
#include <windows.h>
static HINSTANCE hInst;
HINSTANCE getHInst(void)
{
      return hInst;
}

#ifdef WIN32

BOOL WINAPI DllMain(
    IN HINSTANCE hinstDLL,     // handle to DLL module
    IN DWORD fdwReason,        // reason for calling function
    IN LPVOID lpvReserved      // reserved
   )
{
      switch(fdwReason)
      {
            case DLL_PROCESS_ATTACH:
                  hInst=hinstDLL;
                  break;
            case DLL_PROCESS_DETACH:
                  break;
      }
      return TRUE;
}
```

```
#else
int FAR PASCAL LibMain(HANDLE hInstance, WORD wDataSegment,
                       WORD wHeapSize, LPSTR lpszCmdLine)
{
      if (hInst != NULL)
            return FALSE;

      hInst = hInstance;
      return TRUE;
}

int FAR PASCAL __export WEP (int bSystemExit)
{
      hInst = NULL;
      return 0;
}

#endif
#ifndef H450_H
#define H450_H

#include <cm.h>


char tpReadOutVT(HPdlAProtocol haProtocol, HPVT valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab);
char tpPopulateVT (char * dummy, char * dummypath, HPVT hVal, int
nodeId);

#endif /* H450_H *//
```